Parallel Performance of ADPDIS3D - A High Order Multiblock Overlapping Grid Solver for Hypersonic Turbulence

B. Sjogreen, H. C.Yee, J. Djomehri, A. Lazanoff, W. D. Henshaw

October 1, 2009

21st International Conference on Paralell CFD
Moffett Field, CA, United States
May 18, 2009 through May 22, 2009

ABSTRACT

The goal of this paper is to evaluate the parallel performance of a newly developed fluid dynamic flow solver ADPDIS3D. ADPDIS3D is a 3-D variable high order multiblock overlapping grid code in curvilinear geometries. It includes a unified treatment of gas dynamics/MHD (magnetohydrodynamics), multifluid, combustion and nonequilibrium flows. The code is based on low dissipation high-order accurate spatial finite difference methods (Yee et al., 1999; Yee & Sjögreen, 2007, 2008) for turbulence with shock computations. Flow sensors are used in an adaptive procedure to analyze the computed flow data and indicate the amount, location and type of built-in shock-capturing numerical dissipation that can be eliminated or further reduced. By design, the flow sensors, spatial base schemes and nonlinear dissipation models are standalone modules. The current version of the code consists of high order central spatial base schemes of order up to 14, and adaptive nonlinear filters of order up to 9. The code also includes the sixth-order central compact scheme with an eighth-order compact filter. Standard shock-capturing schemes and hybrid schemes of order up to nine are included. To further minimize the use of numerical dissipation, the conservative and non-conservative skew-symmetric splitting of the gas dynamics equations (Sjogreen & Yee, 2009; Yee et al., 2000) are included in the code. ADPDIS3D was originally designed for time-accurate simulation of hypersonic turbulent flows, including combustion, plasma, thermal and chemical nonequilibrium flows. Different type and order of spatial base schemes and filter dissipations can be used on different grid blocks.

ADPDIS3D makes use of overset grids, generated by the grid generator Ogen (Henshaw, 1998). In a parallel performance study, ADPDIS3D shows good weak and strong scaling with up to 15,000 processors on the NASA supercomputer Pleiades. When the number of processors becomes very large, I/O operations can become a critical factor in the overall execution time. We present an improved algorithm for parallel I/O, used by ADPDIS3D, which has good scaling properties.

B. Sjögreen, W.D. Henshaw, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, CA94550, U.S.A
H.C.Yee, J.Djomehri, A. Lazanoff, NASA Ames Research Center, Moffett Field, CA94035, U.S.A.

PARALLEL FLOW SOLVER AND GRID GENERATOR

ADPDIS3D is a high order time-accurate solver that is capable of solving compressible fluids containing a wide range of flow speeds and a number of different governing equation sets. The currently implemented models include the Euler/Navier-Stokes equations of standard compressible gas dynamics, the equations of non-ideal magnetohydrodynamics, and the equations of multi-species combustion with or without thermodynamic non-equilibrium models. The code is especially suitable for simulation of hypersonic turbulent flows, including combustion, plasma, thermal and chemical non-equilibrium flows. ADPDIS3D approximates the flow equations by node centered finite difference discretizations on curvilinear grids. For simulation in complex geometries, ADPDIS3D uses composite overset (multiblock overlapping) grids generated by the Ogen [3] grid generator. Different type and order of spatial schemes can be used on different grid blocks.

ADPDIS3D is based on efficient low dissipative high-order accurate spatial methods that include limiting and filtering with flow sensors [11, 6, 12]. The idea of the method is to advance in time with a highly accurate base scheme, and to apply a non-linear filter after each full time step to suppress Gibbs' oscillations at discontinuities and other unphysical phenomena caused by locally non-smooth solution features. The filters consist of the dissipative portion of a high order shock-capturing method multiplied by a flow sensor, where the flow sensor switches on the dissipation only where needed. Any dissipative portion of a shock-capturing scheme can be used as the nonlinear filter dissipation. By design, the flow sensors, spatial base schemes and nonlinear dissipation models are standalone modules. Therefore, a whole class of low dissipative high order filter schemes can be derived with ease. Unlike standard shock-capturing and/or hybrid shock-capturing methods, the nonlinear filter method requires one Riemann solver per dimension, independent of time discretizations. Thus the nonlinear filter method is more efficient and accurate than its shock-capturing counterparts employing the same order of the respective schemes. The preferred flow sensors in ADPDIS3D are based on a wavelet decomposition of the flow field.

The current version of ADPDIS3D contains central spatial base schemes of orders up to 14, and adaptive nonlinear filters obtained from the dissipative portion of WENO schemes of orders up to nine. The code also includes a sixth-order central compact scheme with an eighth-order compact filter. For cross comparison purpose, standard shock-capturing schemes and hybrid schemes of order one to nine are included. To further minimize the use of numerical dissipation and tuning of filter parameters, the conservative and non-conservative skew-symmetric splitting of the gas dynamics equations [10, 8, 2, 13], and a flow speed indicator [13] are included in the code. Numerical experiments on several dozens of multi-dimensional test cases with a wide range of flow types indicate highly time-accurate and efficient simulations can be obtained.

ADPDIS3D advances the solution in time by Runge-Kutta time stepping, where temporal order of accuracy can be one through fourth-order.two, three. For reacting flows, point implicit and implicit Runge-Kutta methods are included.

One of the complications for parallel computation with high order spatial schemes is that the computational stencils are very wide, which leads to a large number of additional ghost points at the boundaries between processor blocks. Furthermore, ADPDIS3D uses summation-by-parts boundary modification of the operators near the physical boundaries [5, 9]. When the number of processors increases, these boundary operators can make up a significant part of the computational domain in some of the processors. The load balancing algorithm used by ADPDIS3D takes these effects into account.

For parallel execution on overset grids, each component grid is evenly distributed over the total number of processors available. This gives perfect load balancing, but the amount of communication is larger than optimal. For the explicit time stepping used in ADPDIS3D, the communication cost is still only a small fraction of the total computation time. The approach is most efficient when the composite grid is made up of a few large component grids, because of low computation to communication ratio when component grids with very few grid points are distributed on a large number of processors. See the following section for examples of performance.

The overlapping grid generator Ogen (part of the Overture framework developed at Lawrence Livermore National Laboratories) is used to construct overlapping grids for ADPDIS3D. Given a set of structured curvilinear component meshes that cover a computational domain and overlap where they meet each other, the overlapping grid generation algorithm will determine how the grids should interpolate from one another. This process involves the cutting of holes where portions of grids are removed where they lie outside the computational domain. There are a number of important issues that must be addressed to make the grid generation process general, robust and automatic. The algorithm used by Ogen has been recently enhanced to run on distributed memory parallel computers. This speeds up the grid generation process and also permits very large grids to be generated.

The objective of this paper is to evaluate the parallel performance of the newly developed compressible flow solver, ADPDIS3D, by reporting on recently performed benchmark computations with the solver on the new NASA supercomputer Pleiades. The essential feature of ADPDIS3D is that it combines the capabilities of a hypersonic flow solver for complex geometries with the capabilities of a solver for direct numerical simulation of turbulence.

EXAMPLE COMPUTATIONS

Often, hypersonic turbulent flows around re-entry space vehicles and in space physics involve mixed steady strong shocks and turbulence with unsteady shocklets. Figure 1 illustrates a schematic of many of the possible steady and unsteady flow types. While sixth-order or higher order shock-capturing methods are appropriate for unsteady turbulence with shocklets, lower order shock-capturing methods are more effective for strong steady or nearly steady shocks in terms of convergence. In order to minimize the shortcomings of low order and high order shock-capturing schemes, ADPDIS3D allows different order of accuracy on different component grids. The two- and three-dimensional test cases reported in [7] illustrate that the overall error in high speed flow computations is reduced, even if high order schemes are used on only some of the component grids. The two shown in Figs. 2–4 demonstrate the unique capability of ADPDIS3D to both solve for flows in complex geometries, and perform direct numerical simulations of turbulence.

The first example is a 3-D inviscid flow past an Apollo-like re-entry space vehicle with an overlapping grid system indicated in Fig.. 2a. There are six grids in total. The Cartesian background grid has a fairly coarse grid spacing. The body is defined by a spline curve, rotated around the *x*-axis with two orthographic cap grids that cover the

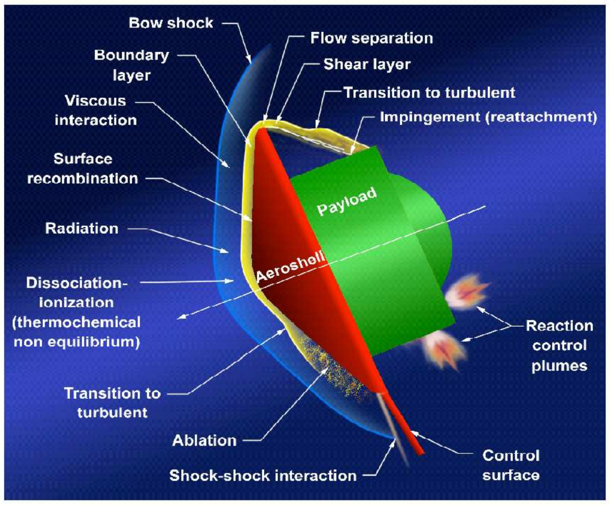

Figure 1. Flow phenomena encountered around an entry vehicle.

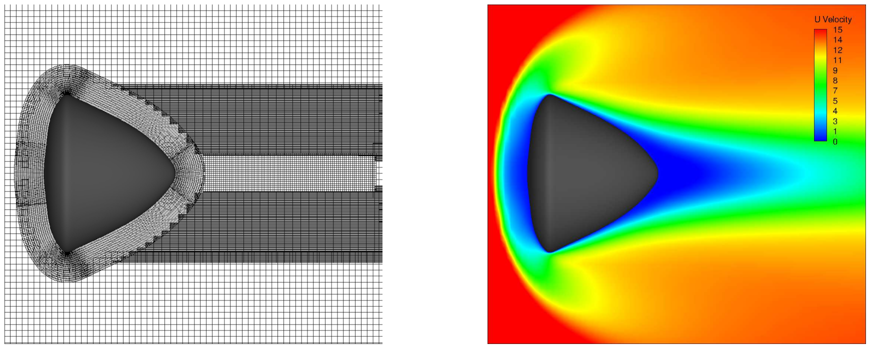

Figure 2. Two-dimensional slice through the grid and Mach number color levels in logarithmic scale for an Apollo-like CEV, Mach 16.

polar singularities. A cylindrical grid together with a Cartesian grid that covers its polar singularity are inserted in the wake region. The grid spacing is approximately 0.05 on the body and wake grids and about five times larger on the Cartesian background grid. Fig. 2b shows computed Mach number color levels on the same 2-D slice as the grid. The free stream Mach number is 16 in this computation.

Figures 4 and 3 show turbulent mixing by a Richtmyer-Meshkov instability [4]. The initial condition is a tube with air to the left and SF_6 to the right of an interface. Figure 3 shows the initial flow configuration. The tube is closed at the right end. A shock wave is sent in from the left. After passing through the air/SF_6 interface the shock wave is reflected at the end of the tube. The reflected shock passes through the interface and mixes the interface once more. Figure 4 shows the interface, represented as the iso-surface of $y_{air} = 1/2$, where y_{air} is the mass fraction of air in the mixture. The interface is given a small initial perturbation to trigger multi-dimensional effects. The increased mixing of the reflected shock can be assessed from Fig. 4, where the left subfigure shows the interface after the first shock interaction, and the right subfigure

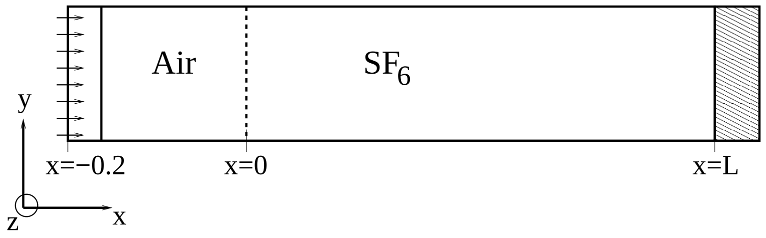

Figure 3. Richtmyer-Meshkov instability, geometry and initial data.

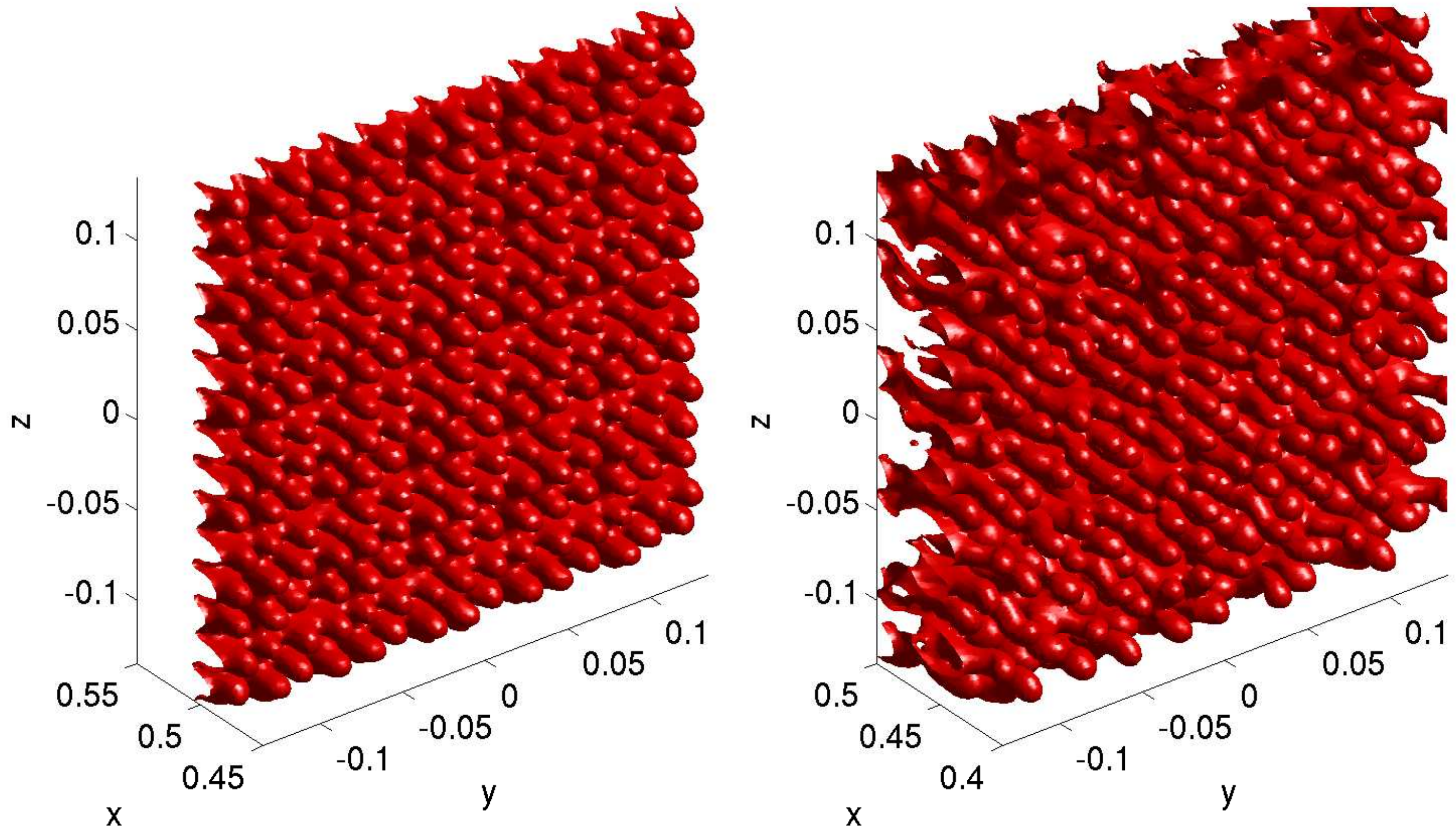

Figure 4. Richtmyer-Meshkov instability in Air/SF_6. Material interface before (left) and after (right) reshock. Iso-surface at 1/2 of air mass fraction.

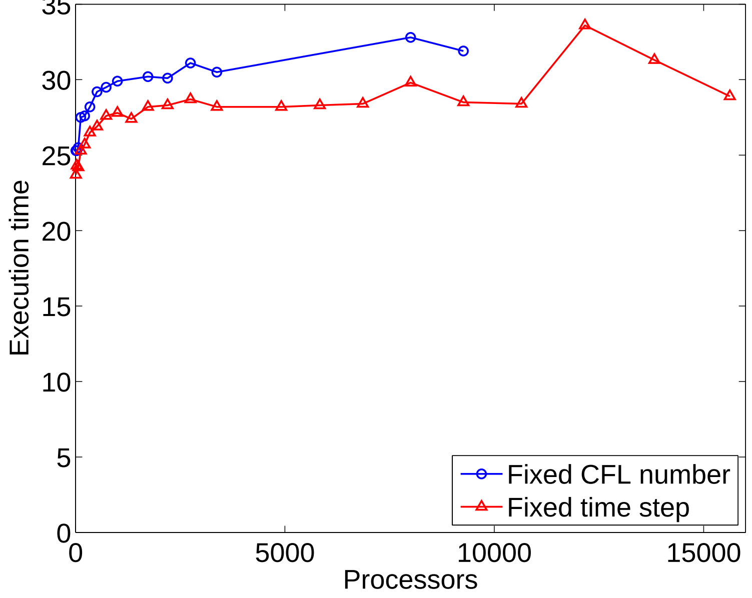

Figure 5. Execution time with weak scaling for a single grid computation for 20 time steps. Fixed time step (red) requires no global communication. Fixed CFL number (blue) computes the local time step and takes global minimum to compute.

shows the interface after the interaction with the reflected shock.

PARALLEL PERFORMANCE

Figure 5 shows weak scaling on Pleiades (A supercomputer with $48,000$ processors at NASA Ames Research Center) for the Taylor-Green vortex problem [1] in three space dimensions on a single block grid. The total execution time for a small number of time steps is measured when the number of grid points per processor is fixed at $216,000$. The number of processors varies up to $15,625$. The number of processors for the data points in Fig. 5 is always a cube, p^3, for integers $p = 2,\ldots,25$. This means that not only is the number of points per processor constant, but also that the shape of data in each processor is always the same (a cube). The work load at the boundaries of the domain is identical in each processor, because the boundary conditions are periodic. Therefore, the number of arithmetic operations is exactly the same in each processor. Any deviation from a perfectly constant execution time must be attributed to differences in network performance. It is not surprising that the performance is superior when the number of processors is small, because, in that case, the processors are closer to each other, in the same node, or in the same rack. Figure 5 shows both the performance with fixed time step (red) and with fixed CFL number (blue). When the time step is fixed, no global communication is necessary, but when the CFL number is fixed, the size of the time step is computed before each time step. This entails more arithmetic operations and a global communication step to transmit the maximum stable time step to all processors. However, as shown by Fig. 5, the cost of the global communication, which theoretically should increase logarithmically with the number of processors, is not significant, at least up to $10,000$ processors.

Figure 6 shows strong scaling on Pleiades for the Apollo-like CEV geometry in Fig. 2, with 65 million grid points on six component grids. Figure 6 shows results

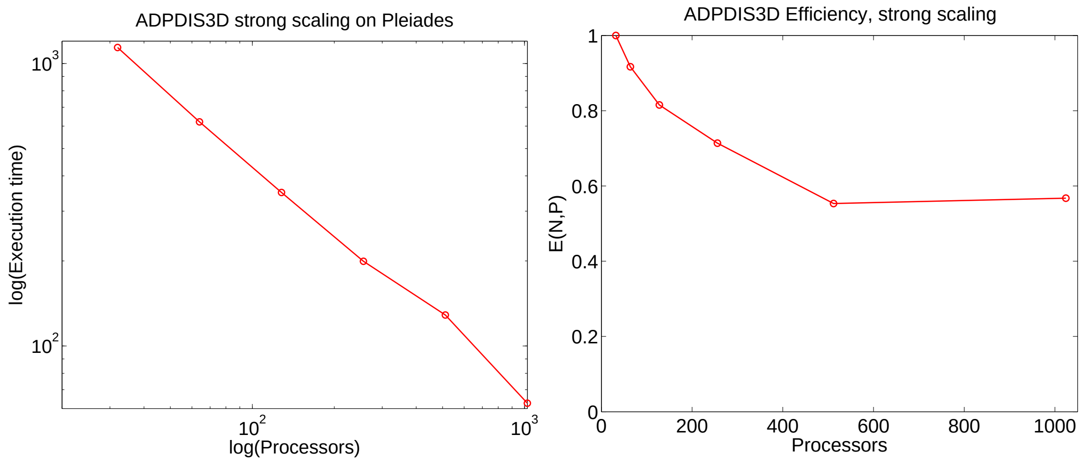

Figure 6. Strong scaling performance for an overset grid system of 60 million grid points. a) (left) Execution time in seconds for 100 steps, logarithmic scale. b) (right) Efficiency (right) for the same data.

with up to 1024 processors, which corresponds to approximately $58,000$ points per processor. The performance started to seriously degrade for larger number of processors. Figure 6a shows the execution time for 100 time steps vs. number of processors, and Fig. 6b displays the corresponding efficiency. The efficiency is normalized to be one at the first data point, which was obtained with 32 processors. Here the nontrivial geometry, grids with cut out holes, and the global communication step required for the interpolation between overset grids are difficulties that are not present in the previous weak scaling example.

PARALLEL I/O

Arrays in ADPDIS3D are distributed on all available processors. An array of $N_1 \times N_2 \times N_3$ points, $u_{i,j,k}$, $1 \le i \le N_1$, $1 \le j \le N_2$, $1 \le k \le N_3$, is distributed on the processors in blocks, where the block assigned to processor p is the sub array $u_{i,j,k}$, $b_{p1} \le i \le e_{p1}$, $b_{p2} \le j \le e_{p2}$, $b_{p3} \le k \le e_{p3}$. The block index intervals, $[b_{p1}, e_{p1}]$, $[b_{p2}, e_{p2}]$, and $[b_{p3}, e_{p3}]$, cover the global index interval, $[1, N_1]$, $[1, N_2]$, and $[1, N_3]$, respectively, with some overlap between processors that depends on the width of the computational stencils used. The sizes of the array block in processor p are denoted $n_{1p} = e_{p1} - b_{p1} + 1$, $n_{2p} = e_{p2} - b_{p2} + 1$, and $n_{3p} = e_{p3} - b_{p3} + 1$.

The array when written to a file, is ordered according to the global index, i.e., the qth element in the file is $u_{i,j,k}$ where $q = i + N_1(j-1) + N_1N_2(k-1)$. The representation in the file does not depend on the number of processors used to write the file.

Previous versions of ADPDIS3D wrote arrays to disk by letting each processor write its own part of the array. The data held by a processor is in general not consecutive data items on the file. Thus the pth processor had to write, in the general case, $n_{2p}n_{3p}$ data chunks, each of size n_{1p}. This algorithm is simple and reliable and works fine with number of processors in the hundreds. However the algorithm turned out to be very inefficient on supercomputers with a very large number of processors. The inefficiencies come both from the large number of processors participating in the writing, and from the large number of write statements each process has to issue.

A recent improvement in ADPDIS3D is an algorithm in which only some of the processors write and where each write operation writes a larger block of data. This is accomplished by ordering data more optimally for I/O by passing it around in the message passing network before each write operation. For ease of presentation, the description of the algorithm below assumes that all integer divisions can be done without remainder. The actual implementation in ADPDIS3D can handle more general dimension sizes.

Assume that the parallel computer has P processors, enumerated $p = 1,\dots,P$. The new algorithm takes as input a list of writing processors, r_q, $q = 1,\dots,Q$, with $Q \leq P$, and a maximum size s, such that an array of size sN_1N_2 can fit into the memory of a single processor. The array is divided into Q slices, with each slice of size sN_1N_2, along the k direction. The number of slices is then N_3/s. Since these slices will be written by Q processors, each writing processor will be responsible for $N_3/(Qs)$ slices. The writing, therefore, takes place in $B = N_3/(Qs)$ steps. In each step a slice of size sN_1N_2 is written from all Q writing processors. These steps will be enumerated $b = 1,\dots,B$. It it straightforward to see that writer q will write the block

$$D_{q,b} = [1,N_1] \times [1,N_2] \times [k_{q,b}+1, k_{q,b}+s]$$

where $k_{q,b} = ((q-1)B+b-1)s$ in step b of the write operation. Before each of these write steps, each processor must send the part of the array slice $D_{q,b}$ that it owns to the writing processor r_q.

Figure 7 displays an outline of the writing algorithm. In Fig. 7, n_b denotes the number of send operations the processor executing the algorithm, performs in write step b, $w_{i,b}$, $i = 1,\dots,n_b$ denotes the processor numbers that the processor executing the algorithm, sends to in step b, and $G_{i,b}$, $i = 1,\dots,n_b$ denotes the intersection of the part of the array that the processor executing the algorithm, holds with the slice that is being written by processor $w_{i,b}$ in step b. If the processor executing the algorithm is a writer processor, the additional data m_b, $y_{i,b}$, and $H_{i,b}$ are needed, where m_b denotes the number of receive operations the processor executing the algorithm, performs in write step b, $y_{i,b}$, $i = 1,\dots,m_b$ denotes the processor numbers that the processor executing the algorithm receieves from in step b, and $H_{i,b}$, $i = 1,\dots,m_b$ denotes the intersection of the part of the array in processor $y_{i,b}$ with the slice that the processor executing the algorithm is writing in step b.

The algorithm displayed in Fig. 8 shows how the data n_b, $w_{i,b}$, $G_{i,b}$ and m_b, $y_{i,b}$, $H_{i,b}$ can be computed. The algorithm in Fig. 8 has one loop that extends over all processors, but the work performed inside the loop is very small. The algorithm also needs to store one integer array of size P. However, this array is not needed once the data structure is set up. If the data structure is computed once and stored before the actual numerical computation, the memory requirement will not grow excessively with P. Scaling to computers with a large number of processors should therefore be reasonably efficient. Parallel file systems allow the processors to write data simultaneously. To allow several processors to write to the same file at the same time, files on parallel file systems are distributed cyclically onto a number of disks. The term 'disk' will here refer to a logical disk, which in reality can consist one or more physical disks, for redundancy and improved performance. This means that if the number of disks ('stripes') is o and the stripe size is s bytes, then the first s bytes of the file is stored on the first disk. The next s bytes are stored on the second disk, etc. up to the oth disk, after which distribution continues with disk number one again. See Fig. 9 for a schematic view.

```
if I am $r_q$ then
    set file pointer to position $(i, j, k) = (1, 1, k_{q,1} + 1)$
endif
for $b$ := 1 to $B$
    for $i$ := 1 to $n_b$
        Send sub-block $G_{i,b}$ of my array to processor $w_{i,b}$
    endfor
    if I am writer then
        allocate array $u$ of size $sN_2N_3$
        for $i$ := 1 to $m_b$
            Receive the sub-block $H_{i,b}$ from processor $y_{i,b}$
            insert the $H_{i,b}$ sub-block into $u$
        endfor
        write $u$ to disk
        deallocate $u$
    endif
endfor
```

Figure 7. Parallel write algorithm implemented in ADPDIS3D.

Performance investigations are complicated because there are a large number of parameters that can vary. Among these are the number of disks, the stripe size, the number of processors participating in the I/O, the data size in each write statement, the problem size, and the total number of processors. In addition, a very significant factor is the number of other users performing I/O operations at the same time. In the examples below, an array of size $1000 \times 1000 \times 500$, which gives a file of size 4 Gbyte when written, is distributed on 1024 processors. The parallel file system used for the experiments has a maximum of 60 disks.

¿From Fig. 10 we infer that unless the number of writing processors is very small, it is always more efficient to use a large number of disks. For $o = 20$ and $o = 60$ the performance levels out when the number of writers exceeds o. After 200 writing processors, the time is almost constant, with some increase as the maximum number 1024 is reached.

All of the times reported in Fig. 10 are significantly smaller than with the older approach where all processors wrote their own data patch to the file. With the old algorithm, it could take more than an hour to write a file of a few gigabytes size.

ACKNOWLEDGMENTS

The financial support from the NASA Fundamental Aeronautics (Hypersonic) program for the second author is gratefully acknowledged. The authors thank Nagi Mansour for providing Figure 1. Work by the first and fifth authors performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

1. M.E. Brachet, D.I. Meiron, S.A. Orszag, B.G. Nickel, and R.H. Morf, *Small-Scale Structure of the Taylor-Green Vortex*, J. Fluid Mech., **130** (1983), 411–452.

```
for $b$ := 1 to $B$
    $i$ := 0
    for $q$ := 1 to $Q$
        if my part of the array intersects $D_{q,b}$ then
            $i := i + 1$
            $w_{i,b} := q$
            $G_{i,b}$ := intersection of $D_{q,b}$ and my part of the array
        endif
    endfor
    $n_b := i$
    for $q$ := 1 to $Q$
        Let $i_0$ be such that $w_{i_0,b} = r_q$ or zero if no such $i_0$ exists.
        Gather $i_0$ to processor $r_q$. Denote the resulting vector $i_{0,p}$.
        if $i_0 \neq 0$ then
            Send index bounds $G_{i_0,b}$ to $r_q$
        endif
        if I am $r_q$ then
            $i := 0$
            for $p$ := 1 to $P$
                if $i_{0,p} > 0$ then
                    $i := i + 1$
                    $y_{i,b} := p$
                endif
            endfor
            $m_b := i$
            for $i$ := 1 to $m_b$
                Receive index bounds $H_{i,b}$ from processor $y_{i,b}$
            endfor
        endif
    endfor
endfor
```

Figure 8. Algorithm for setting up the data structures needed by the algorithm in Fig. 7

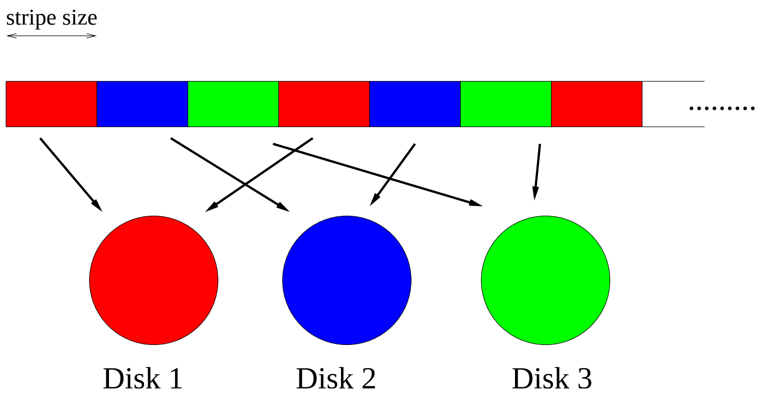

Figure 9. File distributed over a number of physical disks in a parallel file system.

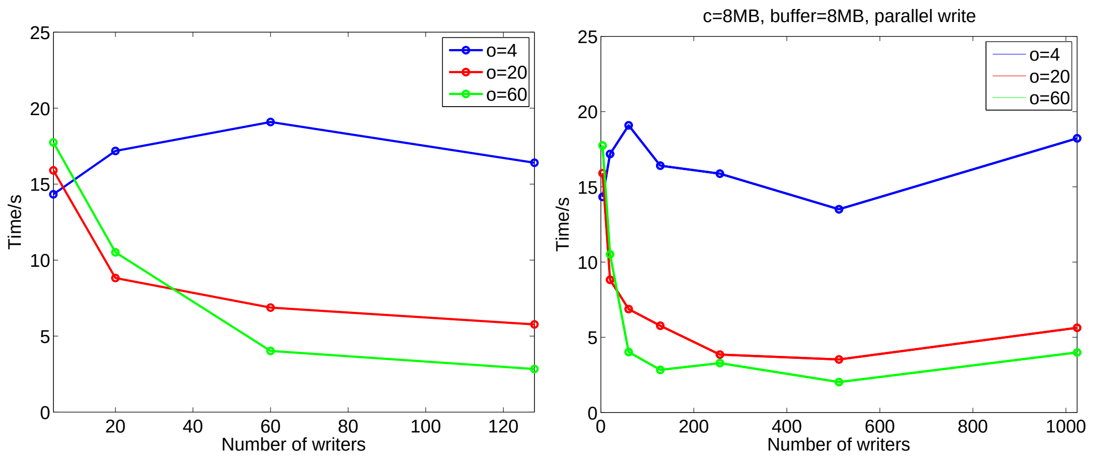

Figure 10. The time in seconds it takes to write a file of size 4 Gbyte distributed on 1024 processors vs. number of writing processors, using the algorithm in Fig. 7. Different curves show different number of disks ('stripes') in the parallel file system. Left subfigure is a close up of the right subfigure.

2. F. Ducros, F. Laporte, T. Souleres, V. Guinot, P. Moinat, B. Caruelle, *High-order Fuxes for Conservative Skew-Symmetric-like Schemes in Structured Meshes: Application to Compressible Fows*, J. Comput. Phys., **161** (2000), 114–139.
3. W.D.Henshaw, *Ogen: An Overlapping Grid Generator for Overture*, Report UCRL-MA-132237, Lawrence Livermore National Laboratory, (1998).
4. D.J. Hill, C. Pantano, and D.I.Pullin, *Large-eddy Simulation and Multiscale Modelling of a Richtmyer-Meshkov Instability with Reshock*, J. Fluid Mech., **557** (2006), 29–61.
5. P. Olsson, *Summation by Parts, Projections and Stability, I*, Math. Comp., **64** (1995), 1035–1065.
6. B. Sjögreen and H. C. Yee, *Multiresolution Wavelet Based Adaptive Numerical Dissipation Control for Shock-Turbulence Computation*, RIACS Technical Report TR01.01, NASA Ames Research Center (Oct 2000); also, J. Scient. Computing, **20** (2004), 211–255.
7. B. Sjögreen and H. C. Yee, *Variable High Order Multiblock Overlapping Grid Methods for Mixed Steady and Unsteady Multiscale Viscous Flows*, Commun. Comput. Phys., **5** (2009), 730–744.
8. B. Sjögreen and H. C. Yee, *On Skew-Symmetric Splitting of the Euler Equations*, Proceedings of the EUNUMATH-09 Conference, June 29 - July 2, 2009.
9. B. Strand, *Summation by Parts for Finite Difference Approximations for d/dx*, J. Comput. Phys., **110** (1994), 47–67.
10. H.C. Yee, M. Vinokur, and M.J. Djomehri, *Entropy Splitting and Numerical Dissipation*, J. Comput. Phys., **162** (2000), 33–81.
11. H.C. Yee and B. Sjögreen, *Development of Low Dissipative High Order Filter Schemes for Multiscale Navier-Stokes/MHD Systems*, J. Comput. Phys., **225** (2007), 910–934.
12. H.C. Yee and B. Sjögreen, *Adaptive Filtering and Limiting in Compact High Order Methods for Multiscale Gas Dynamics and MHD Systems*, Computers Fluids., **37** (2008), 593–619.
13. H.C. Yee and B. Sjögreen, *High Order Filter Scheme for a Wide Range of Flow Speeds*, Proceedings of the ICOSAHOM-09 Conference, June 22-26, 2009, Trondheim, Norway.